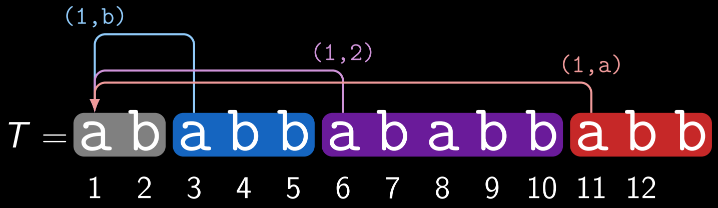
Substring Compression Variants

Dominik Köppl

Faculty of Engineering, University of Yamanashi



coding: (a,b)(1,b)(1,2)(1,a)

setting

text factorization

- input: text T with length n
- output: factorization of T

examples of factorizations

- LZ77
- LZ78
- Lyndon factorization

goal: compute factorization in $\mathcal{O}(n)$ time

substring compression

- index T in a preprocessing step
- query: interval $[i..j] \subset [1..n]$
- output: factorization of $T[i..j]$

goal:

- query time linear to output size (output sensitive)
- index time linear in input size ($\mathcal{O}(n)$ time)

why restricting index time?

trivial solution for substring compression:

- compute and store the factorizations of all $\Theta(n^2)$ substrings
- answer a query in $\mathcal{O}(1)$ via lookup
- however: index space is $\Omega(n^2)$ (hence time is also $\Omega(n^2)$)

work on substring factorization

factorization	construction time	query time	reference
LZ77	$\mathcal{O}(n \lg n)$	$\mathcal{O}(z \lg n \lg\lg n)$	Cormode+'05
LZ77	$\mathcal{O}(n \lg n)$	$\mathcal{O}(z \lg\lg n)$	Keller+'14
Lyndon	$\mathcal{O}(n \lg n)$	$\mathcal{O}(z)$	Babenko+'14
Lyndon	$\mathcal{O}(n)$	$\mathcal{O}(z)$	Kociumaka'16
LZX	$\mathcal{O}(n)$	$\mathcal{O}(z)$	this talk

- z : output size of respective factorization
- $X \in \{$ 78, Miller-Wegman (MW), Double (D) $\}$

References:

- Shibata, K.: "LZ78 Substring Compression with CDAWGs", SPIRE'24
- K.: "Substring Compression Variations and LZ78-Derivates", Information Systems'25
- K.: "Non-Overlapping LZ77 Factorization and LZ78 Substring Compression Queries with Suffix Trees", Algorithms'21

factorizations in this talk

LZ78 derivations

- Lempel–Ziv 78 (LZ78) Ziv,Lempel'78
- Lempel–Ziv Double (LZD) Goto'15
- Lempel–Ziv-Miller–Wegman (LZMW) Miller+'85

why important?

- LZ78 widely used for image compression such as GIF or TIFF
- can be used as a grammar for more operations (unlike LZ77)

why the variants?

- number of LZ78 factors is lower bounded by $\Omega(\sqrt{n})$
- in contrast, the lower bound for LZD and LZMW is $\Omega(\lg n)$

lower bound for LZ78

$$T = \mathtt{a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a} \cdots$$

a	a	a	a	a	a	a	a	a	a	a	a	⋯
1	2	3	4	5	6	7	8	9	10	11	12	

coding:

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZ78

$$T = \texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\,\texttt{a}\cdots$$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZ78

(1,a)

$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a(1,a)

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZ78

(1,a)

(2,a)

$T = \texttt{a a a a a a a a a a a a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a(1,a)(2,a)

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZ78

(1,a) (2,a) (3,a)

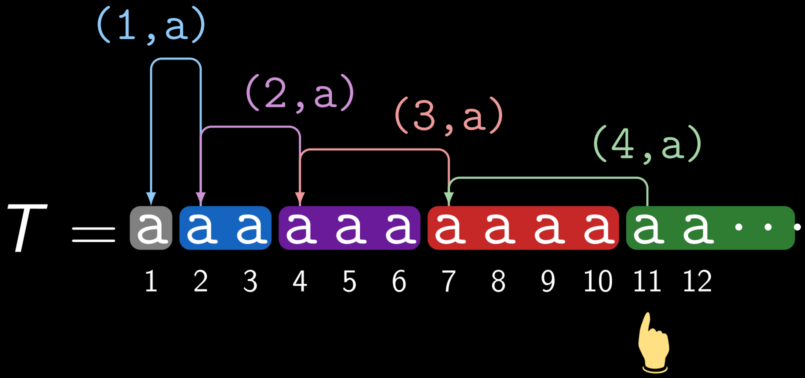
$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: a(1,a)(2,a)(3,a)

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZ78



coding: a(1,a)(2,a)(3,a)(4,a)

since the length $|F_x|$ of the x-th factor is x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\sqrt{n})$

lower bound for LZD

$$T = \mathtt{a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a} \cdots$$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

since the length $|F_x|$ of the x-th factor is 2^x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZD

$T = \texttt{a a a a a a a a a a a a} \cdots$

1	2	3	4	5	6	7	8	9	10	11	12

coding: (a,a)

since the length $|F_x|$ of the x-th factor is 2^x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZD

$(1,1)$

$T = \texttt{a a a a a a a a a a a a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,a) $(1,1)$

since the length $|F_x|$ of the x-th factor is 2^x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZD

(1,1)

(2,2)

$T = $ `a a a a a a a a a a a a ···`

1 2 3 4 5 6 7 8 9 10 11 12

coding: (a,a) (1,1) (2,2)

since the length $|F_x|$ of the x-th factor is 2^x, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW

$T = \mathtt{a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a} \cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding:

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW

$T =$ a a a a a a a a a a a a $\cdots$

a	a	a	a	a	a	a	a	a	a	a	a
1	2	3	4	5	6	7	8	9	10	11	12

coding: a

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW

$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: aa

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW

2

$T =$ a a a a a a a a a a a a $\cdots$

1 2 3 4 5 6 7 8 9 10 11 12

coding: aa2

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW

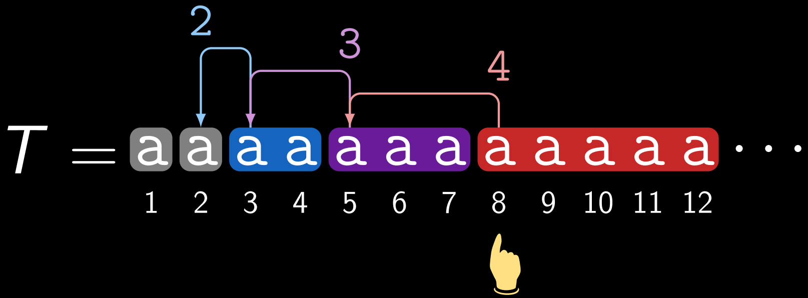
$T = \mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\cdots$

1	2	3	4	5	6	7	8	9	10	11	12

coding: aa23

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number, $\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

lower bound for LZMW



coding: aa234

since the length $|F_x|$ of the x-th factor is the $(x-1)$st Fibonacci number,
$\sum_{x=1}^{z} |F_x| = n \Leftrightarrow z \in \Theta(\lg n)$

definition of LZ78

each factor represented as a pair

- index of a former factor (0 for the empty string)
- appended character

let dst_x denote the starting position of F_x in T.

Definition (LZ78)

A factorization $F_1 \cdots F_z$ of T is LZ78 if

- $F_x = F_y c$, where
- F_y is the longest factor among $F_0, F_1, \ldots, F_{x-1}$ being a prefix of $T[\mathrm{dst}_x..]$,
- $c = T[\mathrm{dst}_x + F_y]$,
- F_0 is the empty string

LZ78

$T =$ a b a b b a b a b b a b b

a	b	a	b	b	a	b	a	b	b	a	b	b
1	2	3	4	5	6	7	8	9	10	11	12	13

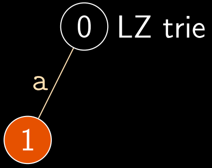
0 LZ trie

coding:

LZ78

$T =$	a	b	a	b	b	a	b	a	b	b	a	b	b
	1	2	3	4	5	6	7	8	9	10	11	12	13

coding: (0,a)

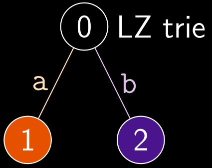


LZ78

$T = $ a b a b b a b a b b a b b

a	b	a	b	b	a	b	a	b	b	a	b	b
1	2	3	4	5	6	7	8	9	10	11	12	13

coding: (0,a)(0,b)

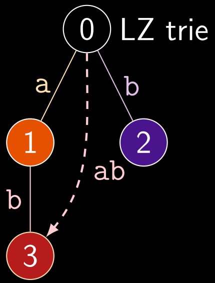


LZ78

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: (0,a)(0,b)(1,b)

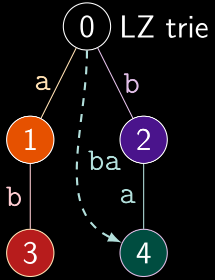


LZ78

$T = \texttt{a}\,\texttt{b}\,\texttt{a}\,\texttt{b}\,\texttt{b}\,\texttt{a}\,\texttt{b}\,\texttt{a}\,\texttt{b}\,\texttt{b}\,\texttt{a}\,\texttt{b}\,\texttt{b}$

a	b	a	b	b	a	b	a	b	b	a	b	b
1	2	3	4	5	6	7	8	9	10	11	12	13

coding: (0,a)(0,b)(1,b)(2,a)



definition of LZD

each factor represented as a pair

- element is either a character or the index of a former factor
- greedily maximize the length by the first element first

let dst_x denote the starting position of F_x in T.

Definition (LZD)

A factorization $F_1 \cdots F_z$ of T is LZD if

- $F_x = G_1 \cdot G_2$ with
- $G_1, G_2 \in \{F_1, \ldots, F_{x-1}\} \cup \Sigma$ such that
- G_1 and G_2 are respectively the longest possible prefixes of $T[\mathrm{dst}_x..]$ and of $T[\mathrm{dst}_x + |G_1|..]$.

example for LZD

$T =$	a	b	a	b	b	a	b	a	b	b	a	b	b
	1	2	3	4	5	6	7	8	9	10	11	12	13

coding:

example for LZD

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: (a,b)

example for LZD

(1,b)

$T = $ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: (a,b)(1,b)

example for LZD

(1,b)

(1,2)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: (a,b)(1,b)(1,2)

example for LZD

(1,b) (1,2) (1,a)

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: (a,b)(1,b)(1,2)(1,a)

definition of LZMW

- has like LZD two references
- however references need to be successive
- thus needs to store only one reference to a former factor index

Definition (LZMW)

A factorization $F_1 \cdots F_z$ of T is LZMW if F_x is the longest prefix of $T[\mathrm{dst}_x..]$ with $F_x \in \{F_{y-1}F_y : y \in [2..\mathrm{dst}_x - 1]\} \cup \Sigma$, for every $x \in [1..z]$.

example for LZMW

$T =$	a	b	a	b	b	a	b	a	b	b	a	b	b
	1	2	3	4	5	6	7	8	9	10	11	12	13

coding:

example for LZMW

$T = \boxed{\mathtt{a}}\,\mathtt{b\,a\,b\,b\,a\,b\,a\,b\,b\,a\,b\,b}$

1	2	3	4	5	6	7	8	9	10	11	12	13

coding: a

example for LZMW

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: ab

example for LZMW

2

$T =$ a b a b b a b a b b a b b

1 2 3 4 5 6 7 8 9 10 11 12 13

coding: ab2

example for LZMW

$T = \texttt{a}\ \texttt{b}\ \texttt{a}\ \texttt{b}\ \texttt{b}\ \texttt{a}\ \texttt{b}\ \texttt{a}\ \texttt{b}\ \texttt{b}\ \texttt{a}\ \texttt{b}\ \texttt{b}$

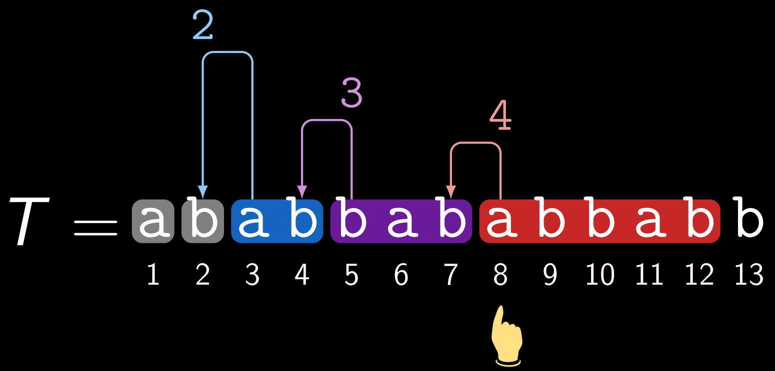
a	b	a	b	b	a	b	a	b	b	a	b	b
1	2	3	4	5	6	7	8	9	10	11	12	13

2

3

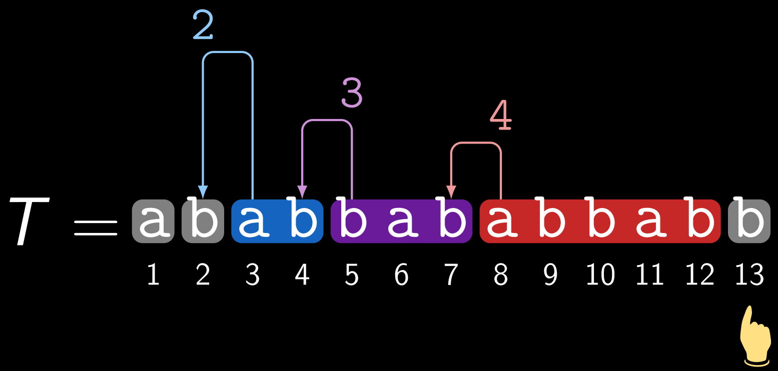
coding: `ab23`

example for LZMW



coding: ab234

example for LZMW



coding: ab234b

LZD and LZMW computation

time	space	reference
$\mathcal{O}(n \lg \sigma)$	$\mathcal{O}(n)$	Goto+'15
$\Omega(n^{5/4})$	$\mathcal{O}(z)$	Goto+'15, Badkobeh+'17
$\mathcal{O}(n + z \lg^2 n)$ expected	$\mathcal{O}(z)$	Badkobeh+'17
$\mathcal{O}(n)$	$\mathcal{O}(n)$	this talk

where

- Goto+'15 only computes LZD
- $\sigma = n^{\mathcal{O}(1)}$, i.e., integer alphabets are supported

For LZ78: $\mathcal{O}(n)$ time and space achieved by Nakashima+'15

our contributions

- for the whole text, we can compute LZD and LZMW in $\mathcal{O}(n)$ time and space
- compute the substring compression of LZD and LZMW with
  - $\mathcal{O}(n)$ index time for preprocessing
  - $\mathcal{O}(z)$ query time
- setting
  - n : length of the input
  - integer alphabet
  - word RAM

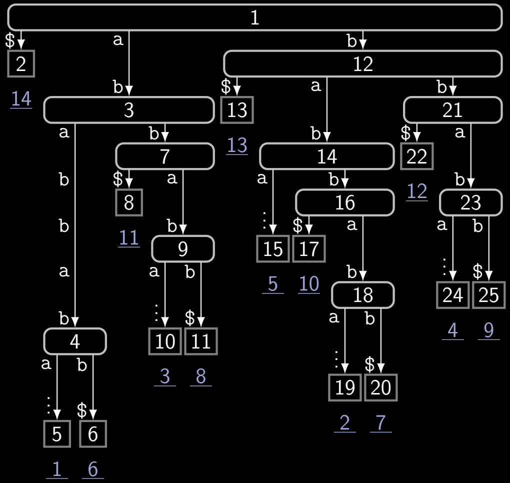
tools

for computation, we leverage the following toolbox

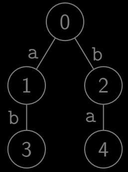
- suffix tree ST Weiner'73
  - linear-time construction of ST Farach-Colton'00
- weighted ancestor query data structure Gawrychowski'14
  - find an ancestor with string depth d of any ST node and any d in $\mathcal{O}(1)$ time
  - constructable in linear time Belazzougui'21
- lowest marked ancestor data structure Cole+'05
  - can mark any ST node in $\mathcal{O}(1)$ time
  - can find the lowest marked ancestor of any ST node in $\mathcal{O}(1)$ time

sum of needed space and time amounts to $\mathcal{O}(n)$ each

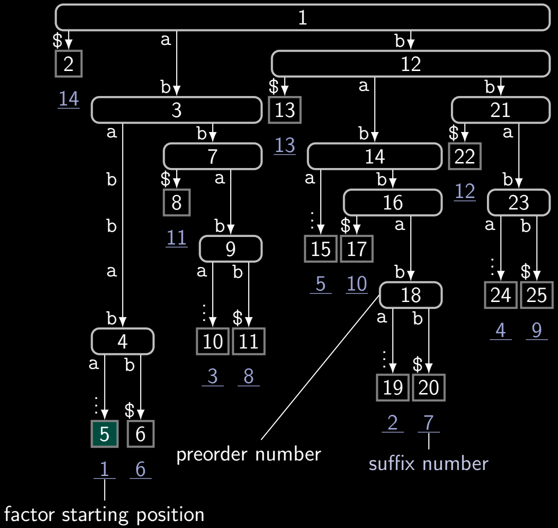
how used for LZ78 computation?

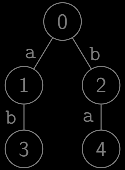


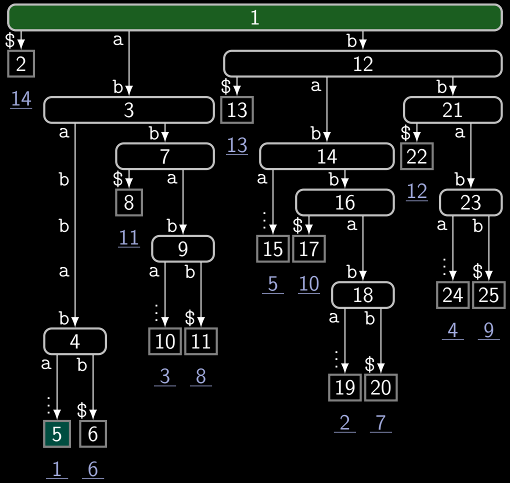
suffix tree of $T\$ =$ ababbababbabb\$



suffix tree of $T\$ =$ ababbababbabb$

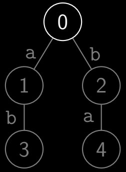


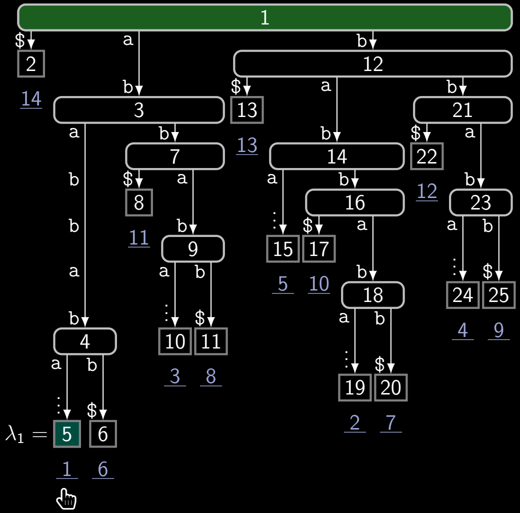




suffix tree of $T\$ =$ ababbababbabb$

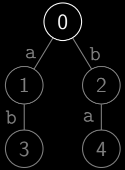
- ST root represents empty factor

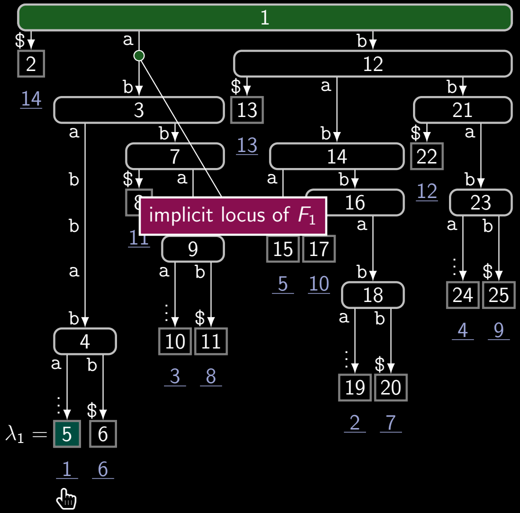




suffix tree of $T\$ =$ ababbababbabb$

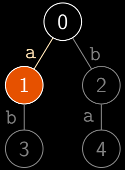
- ST root represents empty factor
- find suffix number = factor starting position

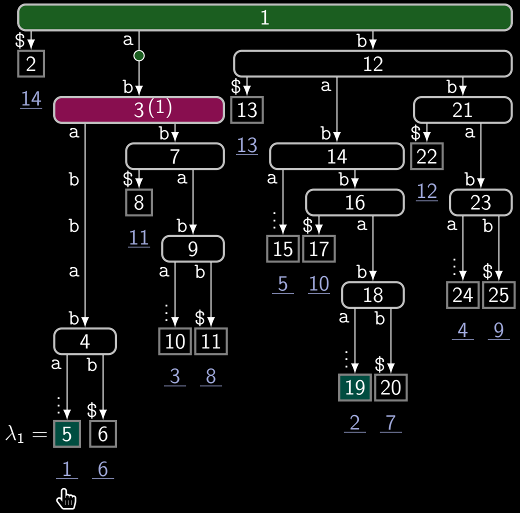




suffix tree of $T\$ =$ ababbababbabb\$

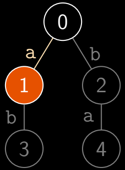
- ST root represents empty factor
- find suffix number = factor starting position
- create child of lowest *marked* ancestor

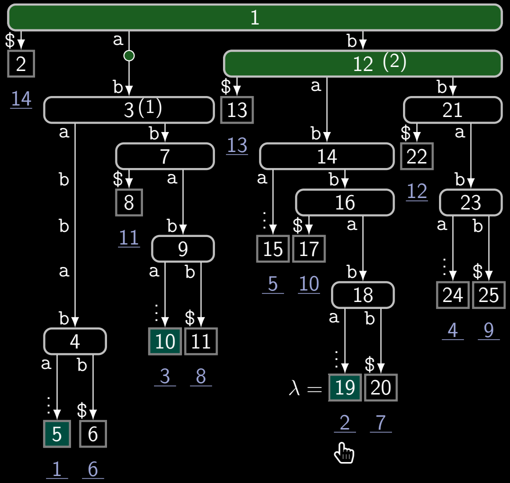




suffix tree of $T\$ =$ ababbababbabb$

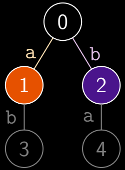
- ST root represents empty factor
- find suffix number = factor starting position
- create child of lowest *marked* ancestor
- explicit nodes witness factors

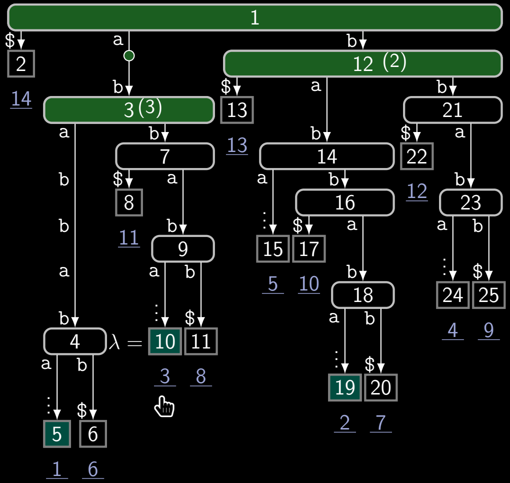




suffix tree of $T\$ =$ ababbababbabb$

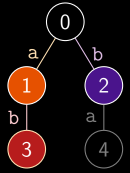
- ST root represents empty factor
- find suffix number = factor starting position
- create child of lowest *marked* ancestor
- explicit nodes witness factors

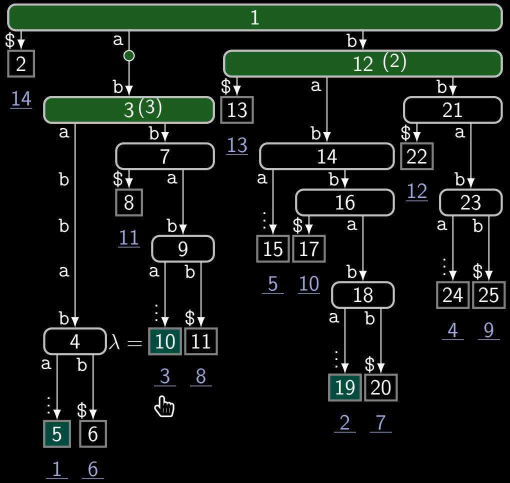




suffix tree of $T\$ =$ ababbabababbabb\$

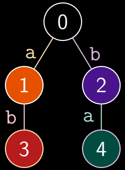
- ST root represents empty factor
- find suffix number = factor starting position
- create child of lowest *marked* ancestor
- explicit nodes witness factors





suffix tree of $T\$ =$ ababbababbabb\$

- ST root represents empty factor
- find suffix number = factor starting position
- create child of lowest *marked* ancestor
- explicit nodes witness factors



time complexity

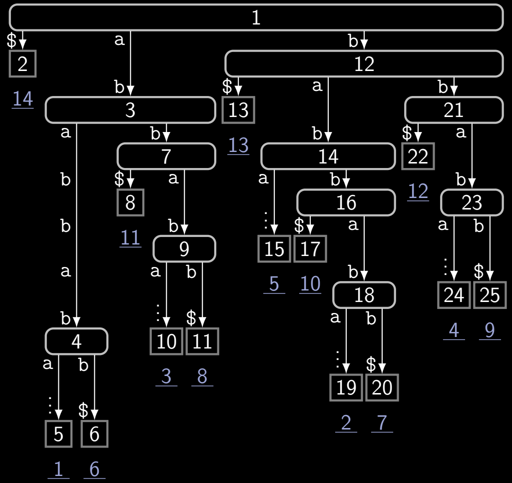
for processing F_x

- take leaf λ corresponding to the starting position dst_x of F_x
- compute the lowest marked ancestor v of λ
- given ℓ is the string length of v, the length of F_x is ℓ
- if v refers to an implicit node, use the stored length instead of ℓ

each step takes $\mathcal{O}(1)$ time, so we have $\mathcal{O}(z)$ total time, where z is the number of processed factors

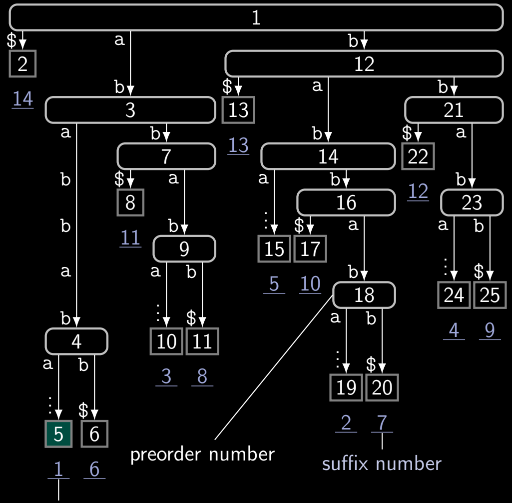
. . . and how about LZD?

suffix tree of $T\$ = \texttt{ababbababbabb\$}$

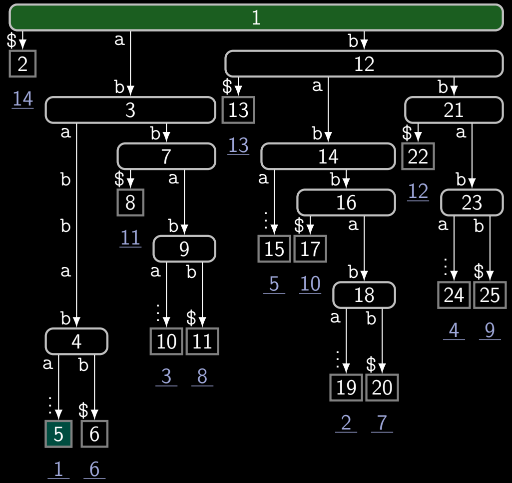


$T = \texttt{ababbababbabb}$

suffix tree of $T\$ = \texttt{ababbababbabb\$}$



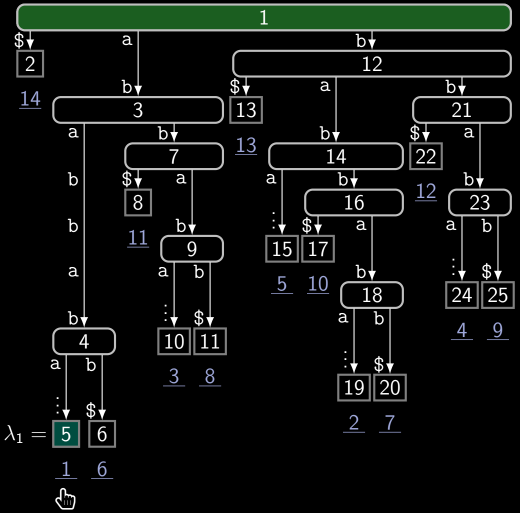
$T = \texttt{ababbababbabb}$



suffix tree of $T\$ =$ ababbababbabb\$

- ST root represents empty factor

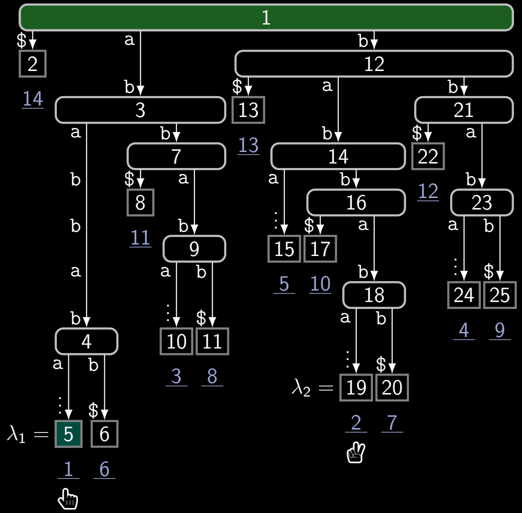
$T =$ ababbababbabb



suffix tree of $T\$ = \mathtt{ababbababbabb\$}$

- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of λ_1 is $\mathrm{dst}_1 = 1$
- lowest marked ancestor of λ_1 is ST root, so $e_L = T[1] = \mathtt{a}$

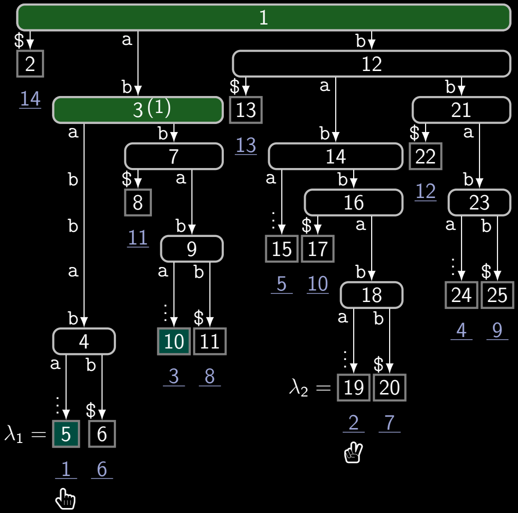
$T = \mathtt{ababbababbabb}$



suffix tree of $T\$ = \mathtt{ababbababbabb\$}$

- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of λ_1 is $\mathrm{dst}_1 = 1$
- lowest marked ancestor of λ_1 is ST root, so $e_L = T[1] = \mathtt{a}$
- λ_2 is leaf with suffix number 2

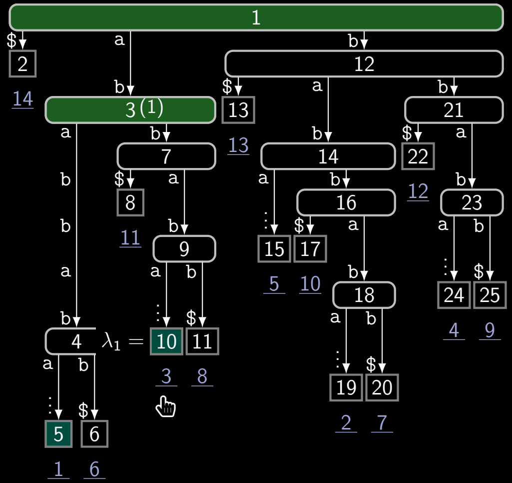
$T = \mathtt{ababbababbabb}$



$T = \text{ab}|\text{abbababbabb}$

suffix tree of $T\$ = \text{ababbababbabb\$}$

- ST root represents empty factor
- compute pair $F_1 = (e_L, e_R)$ of first factor
- suffix number of λ_1 is $\text{dst}_1 = 1$
- lowest marked ancestor of λ_1 is ST root, so $e_L = T[1] = \text{a}$
- λ_2 is leaf with suffix number 2
- lowest marked ancestor of λ_2 is ST root, so $e_R = T[2] = \text{b}$
- mark ancestor of λ_1 with string depth 2 with 1

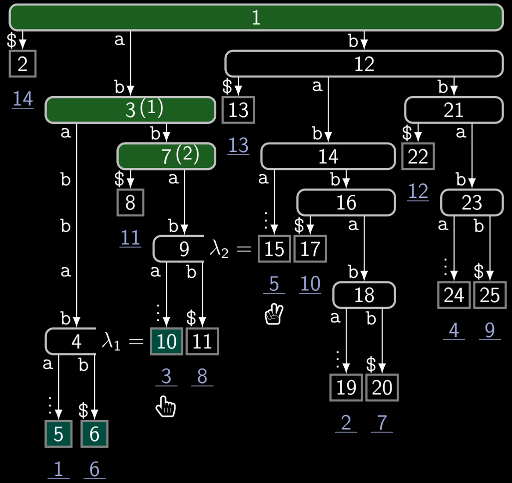


suffix tree of $T\$ = \mathtt{ababbababbabb\$}$

process F_2

- suffix number of λ_1 is $\mathrm{dst}_2 = 3$
- lowest marked ancestor of λ_1 is 3, so $e_L = 1$ (mark of 3)

$T = \mathtt{ab|abbababbabb}$

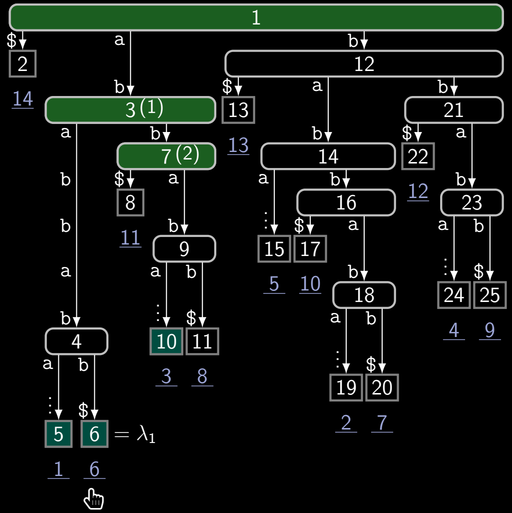


suffix tree of $T\$ = \mathrm{ababbabababbabb\$}$

process F_2

- suffix number of λ_1 is $\mathrm{dst}_2 = 3$
- lowest marked ancestor of λ_1 is 3, so $e_L = 1$ (mark of 3)
- like before, $e_R = T[2] = \mathrm{b}$
- mark ancestor of λ_1 with string depth $|F_2| = 3$ with 2

$T = \mathrm{ab}|\mathrm{abb}|\mathrm{ababbabb}$

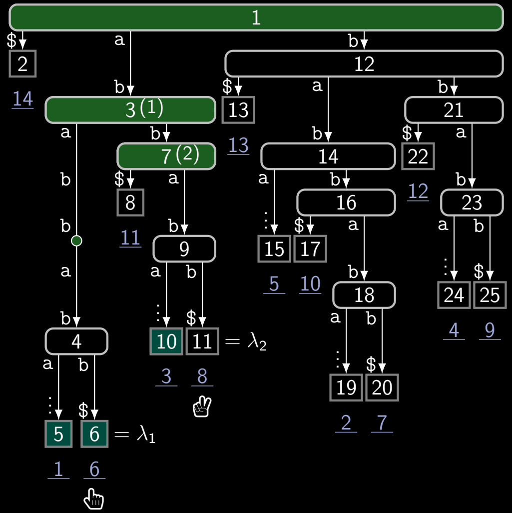


suffix tree of $T\$ = \mathtt{ababbababbbabb\$}$

process F_3

- suffix number of λ_1 is $\mathrm{dst}_3 = 6$
- lowest marked ancestor of λ_1 is 3, so $e_\mathrm{L} = 1$ (mark of 3)

$T = \mathtt{ab}|\mathtt{abb}|\mathtt{ababbabb}$

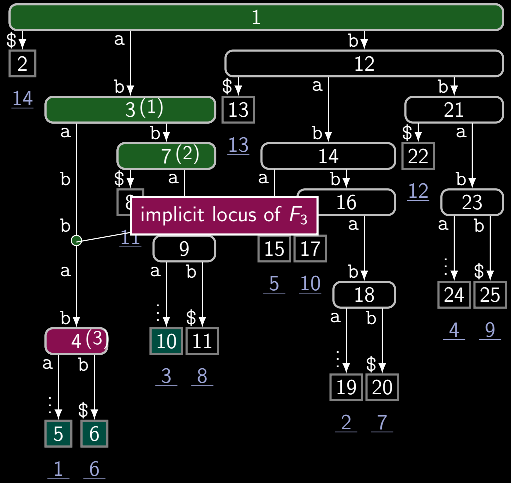


suffix tree of $T\$ = \text{ababbababbabb\$}$

process F_3

- suffix number of λ_1 is $\text{dst}_3 = 6$
- lowest marked ancestor of λ_1 is 3, so $e_L = 1$ (mark of 3)
- lowest marked ancestor of λ_2 is 7, so $e_L = 2$ (mark of 2)
- however: ancestor of λ_1 with string depth $|F_3| = 5$ does not exist!

$T = \text{ab}|\text{abb}|\text{ababb}|\text{abb}$



suffix tree of $T\$ = \texttt{ababbababbabb\$}$

maintaining reference for F_3

- locus of F_3 can be witnesses by node 4
- let node 4 store length of F_3; mark node 4

$T = \texttt{ab}|\texttt{abb}|\texttt{ababb}|\texttt{abb}$

time complexity

basically doubling the time for LZ78

for processing F_x

- take leaf λ_1 corresponding to the starting position dst_x of F_x
- compute the lowest marked ancestor v_1 of λ_1
- given ℓ_1 is the string length of v_1, take leaf λ_2 having suffix number $\mathrm{dst}_x + \ell_1$
- compute the lowest marked ancestor v_2 of λ_2
- length of F_x is $\ell_1 + \ell_2$, where ℓ_2 is the string length of v_2
- if v_1 (or v_2) refers to an implicit node, use the stored length instead of ℓ_1 (or ℓ_2)

each step takes $\mathcal{O}(1)$ time, so we have $\mathcal{O}(z)$ total time, where z is the number of processed factors

LZMW

LZMW computation works similarly

- mark the locus of $F_{x-1}F_x$ instead of F_x
- need only one lowest marked ancestor query (v_2 not needed)

summary

- can compute LZX in $\mathcal{O}(n)$ time, in the computational model
  - n : length of the input
  - alphabet can be integer
  - word RAM

  $X \in \{$ 78, Miller-Wegman (MW), Double (D) $\}$

for substring compression:

- $\mathcal{O}(n)$ index time
- $\mathcal{O}(z)$ query time, where z is the number of factors to output

Open question: Substring compression in compressed space possible?

substring compression in compressed space

- up till now, all substring compression solutions for LZ77, Lyndon factorization, etc., need $\mathcal{O}(n)$ space
- can we improve space by sacrificing time?

Answer

For LZX: Yes by a reduction to the stabbing-max problem

reduction to stabbing-max problem

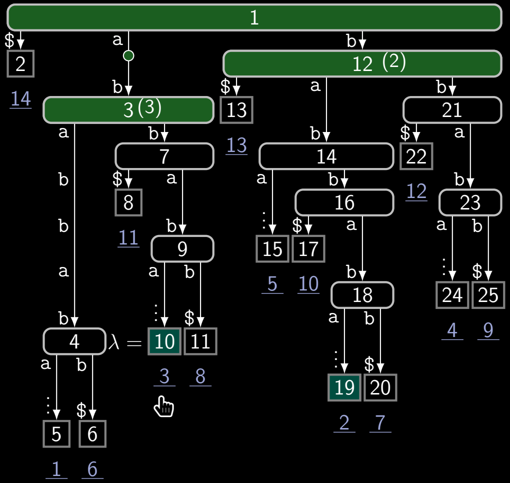
Definition (stabbing-max problem)

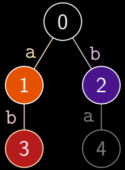
- input: dynamic set of m weighted intervals $\mathcal{S} = \{(\mathcal{I}_i, w_i)\}_i$ with $\mathcal{I}_i \subset [1..n]$ and weight $w_i \in [1..n]$
- supports two operations:
  - $\text{query}(k)$: return the heaviest interval containing k, i.e., $\text{argmax}_i\{w_i \mid k \in \mathcal{I}_i\}$
  - $\text{add}(\mathcal{I}, w)$: add $(\mathcal{I}, w)$ to $\mathcal{S}$

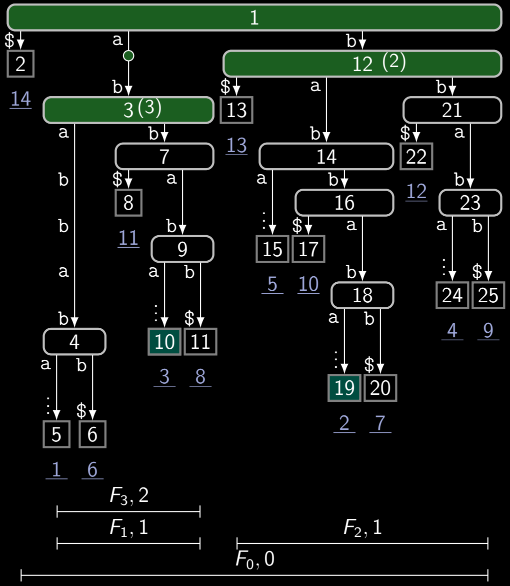
An implementation is due to Yang,Widom'01:

- query and add in $\mathcal{O}(\lg m)$ time
- $\mathcal{O}(m)$ words

LZ78 factorization having F_3 computed

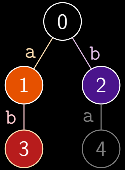


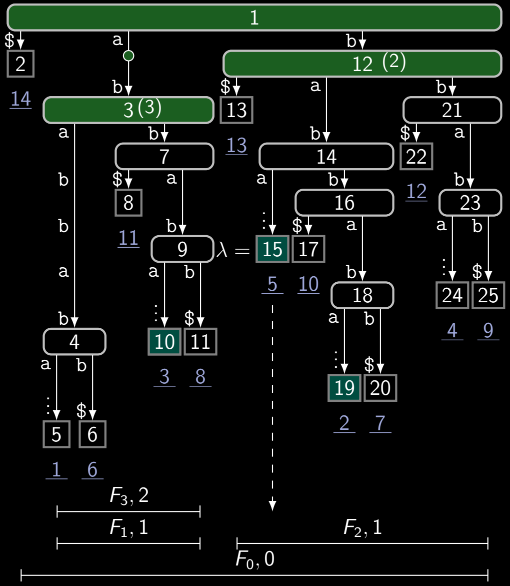




LZ78 factorization having F_3 computed

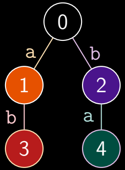
- represent each factor as an interval

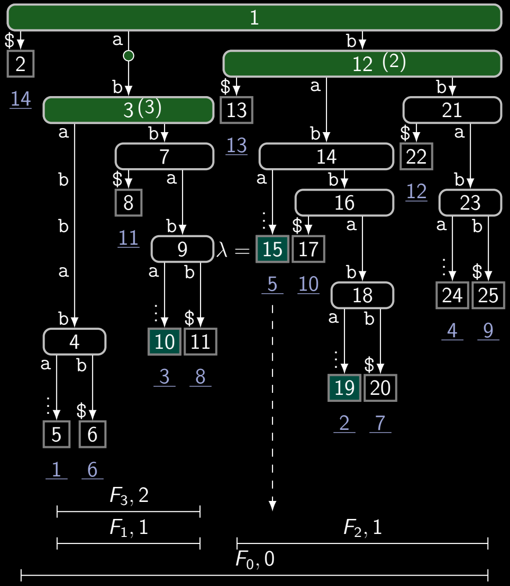




LZ78 factorization having F_3 computed

- represent each factor as an interval
- reference is highest stabbed interval



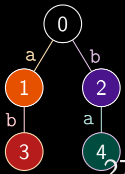


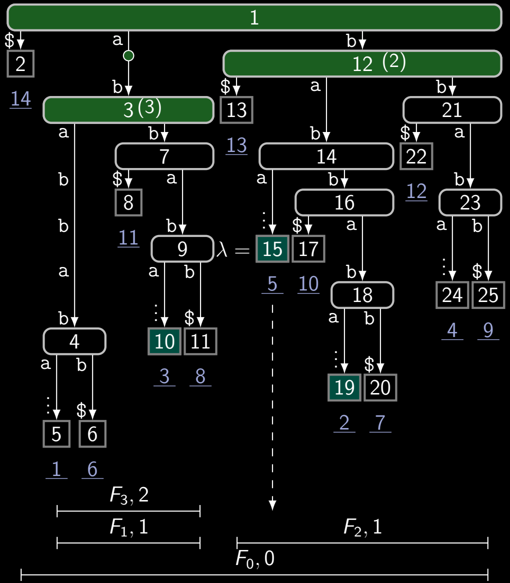
LZ78 factorization having F_3 computed

- represent each factor as an interval
- reference is highest stabbed interval

need two helper arrays LCP[1..n] and ISA[1..n]

- ISA[i]: rank of leaf with suffix number i ($= i$'s SA-position)





LZ78 factorization having F_3 computed

- represent each factor as an interval
- reference is highest stabbed interval

need two helper arrays $\mathrm{LCP}[1..n]$ and $\mathrm{ISA}[1..n]$

- $\mathrm{ISA}[i]$: rank of leaf with suffix number i ($= i$'s SA-position)
- $\mathrm{LCP}[i]$: string depth of lowest common ancestor of i-th leaf with its preceding leaf

the reduction

for a substring Y of T, let $\mathrm{range}(Y) = [i..j]$ be the maximum SA-interval of Y, i.e., Y is a prefix of $T[k..]$ if and only if $\mathrm{ISA}[k] \in [i..j]$.

- represent computed factors by $\mathcal{S}$
- for LZ78: A factor F_x is represented by
  - $\mathcal{I}_x = \mathrm{range}(F_x)$
  - weight $w_x = |F_x|$
- find the next factor starting at $T[i..]$ with $\mathrm{query}(\mathrm{ISA}[i])$:
- if $\mathrm{query}(\mathrm{ISA}[i]) = (\mathcal{I}_x, w_x)$, then F_x is the longest already computed factor being a prefix of $T[i..]$

conclusion

can compute LZ78 with

- stabbing-max data structure
- access to $T[i]$ and $\mathrm{ISA}[j]$ (for factor starting positions)
- $\mathrm{range}(Y)$ for a substring Y (actually only for factors $F_x = Y$ needed)

Implementation: CDAWG

CDAWG Blumer'85

- $\mathcal{O}(e)$ space, where e is the number of edges in the CDAWG
- $T[i]$ and ISA$[j]$ in $\mathcal{O}(\lg n)$ time Belazzougui'17
- range in $\mathcal{O}(\lg n)$ time by centroid-path decomposition Shibata,K'25

Theorem

- *$\mathcal{O}(e)$ space*
- *LZ78 in $\mathcal{O}(z \lg n)$ time with $\mathcal{O}(z)$ extra space*

Implementation: r-index

Define

- $\mathrm{PSV}(x,d) = \max\left(\{0\} \cup \{y \in [1..x-1] \mid \mathsf{LCP}[y] < d\}\right)$ and
- $\mathrm{NSV}(x,d) = \min\left(\{n\} \cup \{y \in [x..n-1] \mid \mathsf{LCP}[y] < d\}\right)$.

Then $\mathrm{range}(Y) = [\mathrm{PSV}(\mathsf{ISA}[i], |Y|), \mathrm{NSV}(\mathsf{ISA}[i], |Y|) - 1]$ for Y being a prefix of $T[i..]$.

r-index Gagie'20

- $T[i]$ and $\mathsf{ISA}[j]$ in $\mathcal{O}(\lg \frac{n}{r})$ time
- PSV and NSV in $\mathcal{O}(\lg\lg \frac{r}{\lg n} + \lg \frac{n}{r})$ time.

Theorem

- *$\mathcal{O}(r \lg \frac{n}{r})$ words of space*
- *LZ78 in $\mathcal{O}(z\left(\lg\lg \frac{r}{\lg n} + \lg \frac{n}{r} + \lg z\right))$ time with $\mathcal{O}(z)$ extra space*

Implementation: δ-index

δ-index Kempa, Kociumaka'23

- $\mathcal{O}(\delta \lg \frac{n \lg \sigma}{\delta \lg n})$ space, where $\delta = \max\{\frac{d_k}{k} \mid k \in [1..n]\}$, and d_k is the number of distinct k-length substrings in T
- $T[i]$ and $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^{4+\varepsilon} n)$ time, where $\varepsilon > 0$ is a given constant.
- longest common prefix between two suffixes in $\mathcal{O}(\lg n)$ time.
- compute $\mathrm{PSV}(x, d)$ and $\mathrm{NSV}(x, d)$ by binary search and LCE queries. (time dwarfed by access to $T[i]$)

Theorem

- $\mathcal{O}(\delta \lg \frac{n \lg \sigma}{\delta \lg n})$ *space*
- *LZ78 in* $\mathcal{O}(z \lg^{4+\varepsilon} n)$ *time with* $\mathcal{O}(z)$ *extra space*

final recap

data structure	space	query time
suffix tree	$\mathcal{O}(n)$	$\mathcal{O}(z)$
CDAWG	$\mathcal{O}(e)$	$\mathcal{O}(z \lg n)$
r-index	$\mathcal{O}(r \lg \frac{n}{r})$	$\mathcal{O}(z \left(\lg \lg \frac{r}{\lg n} + \lg \frac{n}{r} + \lg z\right))$
δ-index	$\mathcal{O}(\delta \lg \frac{n \lg \sigma}{\delta \lg n})$	$\mathcal{O}(z \lg^{4+\varepsilon} n)$

where

- δ: substring complexity
- e: # CDAWG edges
- r: # runs in the BWT
- $\delta \leq r \leq e \leq n$
- z: LZX factors of query substring

Are there other compression methods having whose substring compression can be computed in compressed space?

Thank you for listening. Any questions are welcome!

open problems

- LZ77-based substring compression use geometric data structures, which are heavyweight. Is there some other approach?
- allowing non-greedy choices for LZD/LZMW, the variant computing the fewest factors is NP-hard? (For LZ78, the flexible parsing is optimal)
- Lyndon factorization can be computed with $\mathcal{O}(1)$ space and $\mathcal{O}(n)$ time, so is there likely a trade-off for substring computation?
- "LZSE: an LZ-style compressor supporting $\mathcal{O}(\log n)$ time random access" (arxiv'25) seems to be computable with the presented tools